VIPER Lunar Rover Agile Mission Systems

Jay Trimble
NASA Ames Research Center, Mountain View, CA, 94035, USA

The VIPER Lunar Rover, scheduled for a November, 2024 launch, is a solar powered rover that will search for volatiles at the lunar south pole. VIPER is an operationally complex mission operating in a challenging lighting and communications environment, requiring new design in a number of areas, from mission planning, to real-time waypoint driving. The combination of solar power, limited battery and the dynamic movement of shadows at the south pole gives VIPER small operational margins. The Mission System will be used to operate VIPER during cruise and on the lunar surface. To maximize efficiency and flexibility in Mission System design and thus to improve the performance and reliability of the resulting Mission System, we are tailoring Agile principles that we have used effectively in ground data system software development and applying those principles to the design of elements of the mission system.

I. Introduction

Agile is generally characterized by "quickness, and ease of movement." In software, agile is now well established. We are taking agile methods developed internally for delivery of ground data system software components into mission environments and applying them, with modifications, to more of the mission system.

We define the mission system (MS) to be the combined mission operations system (MOS) and ground data system (GDS). The MOS is the team, operational products and processes needed to operate the mission. The GDS is the hardware, software and facilities used by the MOS to operate the spacecraft.

II. Operating VIPER

An overview of VIPER mission operations are shown in figure 1. The rover is driven in real time in roughly five meter increments called waypoints. There is a six to ten second round trip time delay from sending a command to receiving the results on the ground. Unlike driving on Mars, where a command cycle is created over the course of an Earth-day and uplinked to execute over one or more Martian sols, a VIPER driving command cycle is roughly 300 seconds, with drivers issuing commands at each waypoint. To operate, VIPER requires line of site communication to Earth, as well as Sun on the solar panels. The lighting environment at the lunar south pole is highly dynamic, with shadows moving at roughly the rate of the rover. The mission operates on tight time margins to accomplish the science objectives while operating in both communications and sun line of site and staying ahead of the shadows. The operations design, both for mission planning and real-time operations, is complex and challenging.

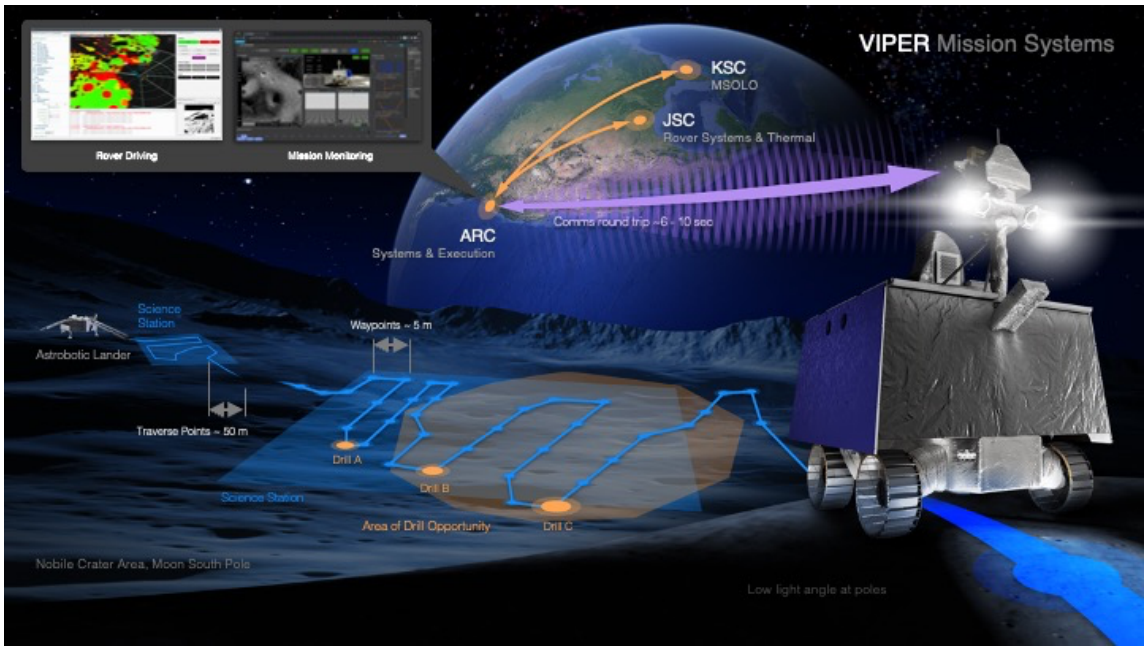

Figure 1 - VIPER Mission System Overview

III. Background: From Waterfall to Agile Software Development

In our previous work with software development for missions[2], we moved from using a traditional waterfall development methodology, to adopting parts of an Agile methodology, then to a user-centered Agile methodology. This evolution was driven by the need to solve problems we encountered in the development process, rather than by a desire to fit a particular development paradigm.

A. Problems with Waterfall in Software

Our group started developing software for missions years before VIPER. In the beginning we had a six month cycle of requirements definition, development, testing and delivery. The fundamental problem was the length of the cycle, which created a large number of requirements to implement over each long cycle. This in turn complicated design, development and testing. Difficult and complex development tasks were sometimes deferred over the long cycle. We used traditional measures such as lines of code as progress metrics. While they were easy to measure internally, we found that these metrics did not translate to meaningful demonstrations of progress either internally or externally. Testing six months of new code at the end of a cycle meant that any bugs found were expensive to fix, and there was always a strong pressure to ship since the customer had been waiting for months. The result was that minor issues were resolved with workarounds, and the customer had to wait another six months for resolution of issues found in acceptance testing.

Six months is a long time from specification to delivery. Over each six month period we fell out of touch with our customers. Our expectations diverged, creating a mismatch that resulted in disappointment and frustration for both the developers and our NASA colleagues to whom we delivered.

B. Steps Toward Agile Software Development

Our first step towards Agile was to shorten our delivery cycle, initially to six weeks, then to three. Per industry standards, each three week develop/test/deliver cycle was called a sprint. Four sprints constituted a release (figure 1). Although this allowed us to find and fix bugs found during unit (development) testing much more efficiently, we found that just shortening the cycle was not enough. Each sprint needed a clear set of objectives, tied to a strategic road map. We set up a three-tiered structure for interaction with our customer. The first was a nightly or continuous build. Every day the customer could download the latest software (figure 2). The measure of progress became what we demonstrated with working code, not what we described on a presentation chart.

The nightly build allowed the customer, and our internal quality assurance team, to try new features as they rolled out, giving us feedback immediately. Each three-week sprint was a deliverable, which the customer put through

testing, and then accepted or rejected features. While sprints were intended for testing and feature acceptance/rejection, a release was intended for deployment into a mission operations environment.

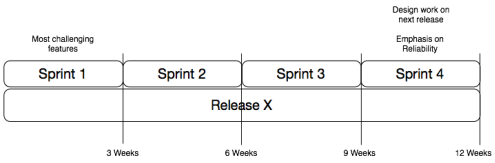

Figure 2 - Four Sprint Release Cycle

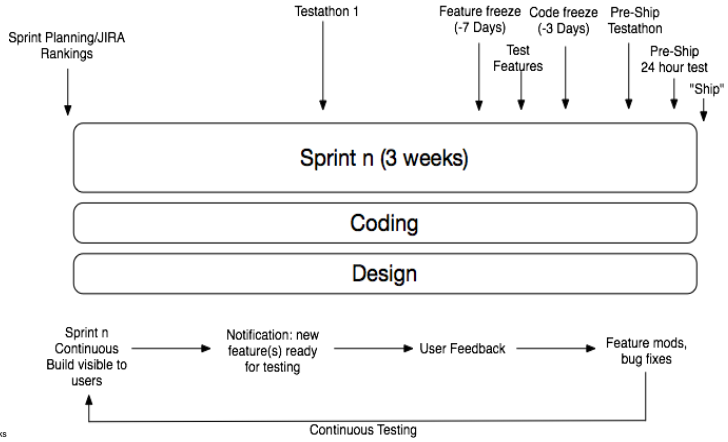

Figure 3 - Agile Sprint Detail

A key principle in this approach is that the schedule of deliveries is predictable and rapid. In effect, the train (the delivery) always goes on time. Features that are ready for delivery make it onto the train. Features that are not ready must wait for the next train. For this to work, the trains must leave regularly and frequently. In other words, sprints must be short and consistent, they must be delivered on time, and features that are not ready to ship may ship in a subsequent sprint. The only reason for delaying a shipment is if the software does not work.

The move to Agile solved our most pressing development and delivery problems. The short delivery cycle created a manageable set of features to develop and test in a bounded period of time. We were able to show demonstrable progress to ourselves and our customer on a frequent basis. We were able to solicit nearly instant feedback on the development of new features. Customer and team engagement and satisfaction went up.

IV. Agile for Mission Systems

In extending agile beyond software, to the MS, we seek benefits similar to those that we saw in software development. Let's compare principles of agile for software, to our tailored agile principles for the MS.

Agile is a broad family of software development methods characterized by rapid iterations and continuous customer feedback. While there are many variations of Agile in practice, they all share common roots in the original Agile manifesto[5]:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiations
- Responding to change over following a plan

An MS analogy to "the measure of progress is working code" is that we measure the capability of the integrated system of people, processes and tools through what we can demonstrate. Before using Agile methods, we assessed the state of a software project with metrics and analysis but could not try the working code until testing and delivery. With modern software methods, we have nightly or continuous builds for code, and we can test capability every day.

There is no nightly build for mission operations systems processes similar to the nightly builds for the software. To demonstrate capability we use different methods, from procedure walkthroughs, to development simulations in a lab, to mission simulations in a control center using the mission software with a simulator.

Tailored agile principles for MS:

- Assessment of capability through demonstrated execution of mission capability
- Maturation, evaluation and iterative development of the system through continuous use during development
- Early and frequent builds and tests

On VIPER MS, in addition to using agile for the GDS, we have applied agile to the heart of MOS development, including, but not limited the following activities:

- Development simulations
- Operational Product Development
- Test and Training

A. Development Simulations and "say it then sim it"

Simulation is a well established method in mission operations for training the operations team. Simulations for training typically begin late in a mission flow, after the system is designed and most of the software is delivered. The purpose is to train the team to fly the mission. We are advancing this flow with development simulations (dev sims). These are conducted early in mission development, as part of the design process. In this phase we can still influence the design of both the space and ground segments.

VIPER dev sims focused on driving. The results influenced the MS and the rover design. A key to making the tight timeline of the VIPER mission is the average speed of the rover over time, considering all the potential factors that influence that average speed, such as time to download images, communications time delay and driver decision time between way points. We call this speed made good (SMG) and it is used in planning the mission traverses. Early estimates of SMG were based on data from dev sims. In those dev sims, we discovered significant issues with aft and sunward driving. This data was used to change the design of the rover aft cam system.

Dev sims provide an early capability to try targeted aspects of mission operations, rather than just talk about them. Note that development of simulation capability is a significant investement of time and resources however, all of those capabilities should feed forward to mission use. The capability of dev sims is both enabled and limited by the maturity of the software at any given time.

"Say it then sim it" refers to a technique we have used in meetings. As with so many space mission teams, the VIPER team has many people with strong opinions. Where those opinions may result in many possible solutions to a problem our solution is, where possible, to conduct a simulation to either resolve the difference or at least to add data to the discussion.

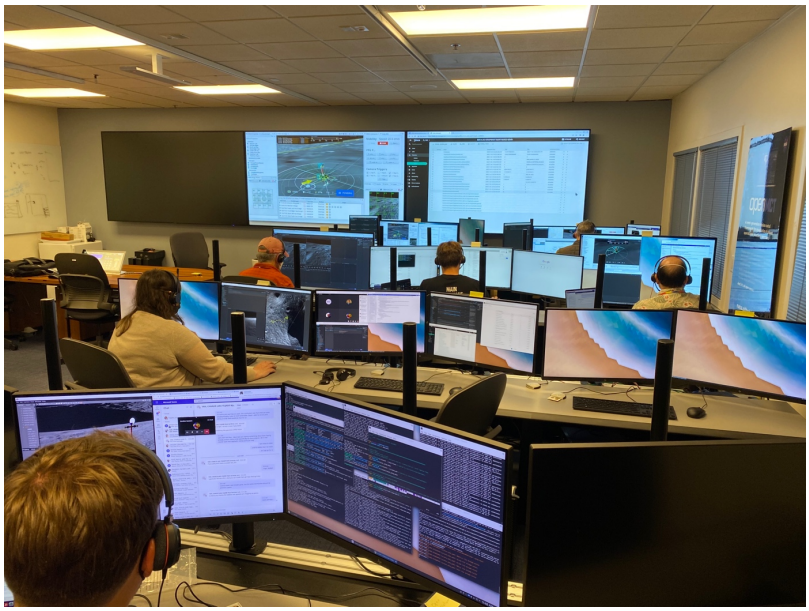

Figure 4 - Dev Sim in the VIPER MS Lab

B. Opertional Product Development

Agile operational product (ops product) development changes both the structure and measure of how we create ops products. The structure is iterative, conducted in sprints similar to software development sprints but driven by the test and training schedule. The measure is what can demonstrate in an environment that allows us to evaluate the utility of the product for the mission, typically a dev sim in the lab or a test and training activity.

Rather than delivering ops products in large drops, as in a waterfall model, we deliver more frequent and smaller drops developed in sprints based on dependencies driven initially by the engineering schedule and then by the test and training simulation schedule. The ops product sprints deliver the capability needed to execute each simulation with the product needs determined by the activities performed in the training/simulation activity which in turn are derived from the concept of operations.

C. Test & Training

Test and Training has the dual role of testing the integrated MS, to show that the people, processes and tools all work together to execute the mission, as well as performing validation on the software stack and system elements. Training an operations team "should" be done with the completed mission software. However, we are building the VIPER rover and MS together and the tight project schedule does not allow for training to begin only after system completion. Early in the test and training program, there is an overlap in training with the late stages of system development, integration and test.

The test and training program is structured in a model of progressively testing and training by executing mission activities based on the concept of operations and the mission timeline and surface traverse plan. Agile elements in test and training include the sprint structure of the development of products to enable the sims as well as the iterations and feedback loops into both the operations software and the ops products. Each test and training activity and associated de-brief feeds data back into the next iteration of software and operational products. Simulations followed by de-briefs and updates to procedures have been done since the days of Apollo and this is well established. In the days of Apollo and the early Space Shuttle, procedure modifications were done quickly, on paper. The difference on VIPER is the incorporation of the sprint structure and the iterative feedback loops ability to quickly modify both ops products and software, all of which is enabled by the flexibility of modern software and development tools.

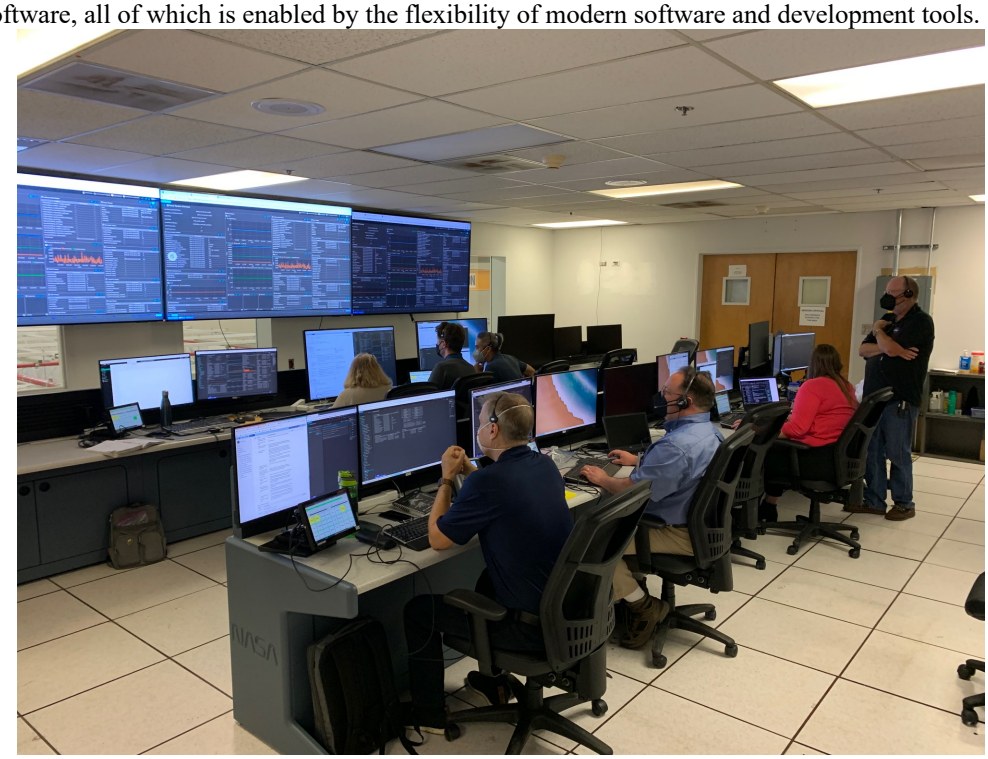

Figure 5 - An early VIPER Mission Simulation, in the Mission Operations Center

V. Integrating Agile into the System Engineering and Requirements Cycle

NASA mandates system engineering processes that vary by mission classification. The implementation of system engineering requirements can vary across the spectrum of missions, from small spacecraft to complex human rated systems. How do agile methods fit within a system engineering process of documents and gate reviews? For VIPER MS gate reviews complement agile methods and serve as an effective means to provide detailed measurements of our overall progress.

The VIPER MS writes requirements documents, down to level four. For software with a human interface we may also write other types of specifications, such as wireframes. Requirements exist at multiple levels of abstraction and specify what we are trying to achieve. Verification processes are designed to test the system against requirements. We also have validation processes which are designed to determine if the system solves the intended problem. These are different methods of testing, with verification being done by a tester who tests the system against requirements while validation is done by the mission operations team in a simulation environment, using the software in a similar manner as we would in flight.

In our experience, attempting complex integrations over the time scale of major system deliveries, typically months, presents several problems, the most significant of which is that the combined hardware and software system may not work, may require significant fixes due to accumulation of problems over time and/or may not work as the customer expects. The problem we encountered was that the each team, rover and MS, developed their systems using agile methods, but we developed separately, and then attempted to integrate only for major deliveries, on the timescale of months. This was an organizational divide and we solved this problem by conducting weekly software integrations across rover and MS so that we would be constantly fixing problems, using the software, and not allowing issues to accumulate. This is a key tenet of agile, continuous integration.

VI. Conclusion

We have taken Agile principles originally evolved for software development and are applying them across the mission system for one NASA mission. The principles apply to both the ground data system and to the mission operations system (people and processes). We are tailoring the principles and focusing on where we believe they will provide the most value, specifically to the unique design challenges of the VIPER mission. We have been able to assess and adjust required flight capabilities unusually early in the life cycle of the project. Further, by deploying and testing with an integrated operations and flight system, we are rapidly maturing the processes and tools. We plan to continue to apply and adapt these principles throughout the project phases, and we expect to continue to realize benefit.

References

[1]Trimble, J. "Lean Mission Operations Systems Design - Applying Lessons from Agile and Lean Software Development to Mission Operations Design", *SpaceOps 2014 Conference*, SpaceOps Conferences, (AIAA 2014-1816). http://dx.doi.org/10.2514/6.2014-1816

[2]Trimble, J. "Agile Development Methods for Space Operations", *SpaceOps 2012 Conference*, SpaceOps Conferences, (AIAA 2012). http://dx.doi.org/10.2514/6.2012-1264554

[3]Marshall, W. et al., 2011. Locating the LCROSS impact craters. *Space Sci. Rev*., 1–22, ISSN:0038-6308

[4]Quinn, J., J. Smith, J. Captain, A. Paz, A. Colaprete, R. Elphic, and K. Zacny. "Resource Prospector: The RESOLVE Payload." USRA Houston. Lunar Exploration Analysis Group (2015), 22 Oct. 2015. Web. 25 Mar. 2016. <http://www.hou.usra.edu/meetings/leag2015/pdf/2046.pdf>.

[5]Beck, Kent et al. "Manifesto For Agile Software Development". *Agilemanifesto.org*. N.p., 2001. Web. 29 Apr. 2015.

Trimble, J, Shirley, M, Hobart, S. “Agile: From Software to Mission System.” SpaceOps Conference 2016